Concretização de um cenário de carros cooperantes num ambiente móvel sem fios

Paulo Sousa
Pedro Martins
António Casimiro
Paulo Veríssimo

DI–FCUL TR–03–23

Julho 2003

Departamento de Informática
Faculdade de Ciências da Universidade de Lisboa
Campo Grande, 1749–016 Lisboa
Portugal

Technical reports are available at `http://www.di.fc.ul.pt/tech-reports`. The files are stored in PDF, with the report number as filename. Alternatively, reports are available by post from the above address.

Concretização de um cenário de carros cooperantes num ambiente móvel sem fios

Paulo Sousa
pjsousa@di.fc.ul.pt
FC/UL*

Pedro Martins
pmartins@di.fc.ul.pt
FC/UL

António Casimiro
casim@di.fc.ul.pt
FC/UL

Paulo Veríssimo
pjv@di.fc.ul.pt
FC/UL

Resumo

Os carros do futuro comunicarão entre si usando ligações sem fios para conseguirem cooperar em certas ocasiões. Esta cooperação tem subjacente requisitos de pontualidade[11, 10]. Infelizmente, é bastante difícil desenvolver aplicações com requisitos de pontualidade em ambientes móveis sem fios ou, mais genericamente, em ambientes de sincronia incerta. O modelo Timely Computing Base (TCB)[12] visa precisamente oferecer uma solução arquitectural que permita conciliar os requisitos de pontualidade das aplicações com os problemas da incerteza do ambiente. Neste artigo é descrito como uma concretização do modelo TCB foi usada na construção de um cenário de carros cooperantes num ambiente móvel sem fios.

Palavras Chave: ambientes sem fios, dispositivos móveis, pontualidade, qualidade de serviço

1 Introdução

Os carros do futuro comunicarão entre si usando ligações sem fios para conseguirem cooperar em certas ocasiões. Por exemplo, ao aproximarem-se de um cruzamento não sinalizado, os carros terão que se coordenar para garantir que não acontecem acidentes. Esta cooperação tem subjacente requisitos de pontualidade, nomeadamente cada carro tem que conhecer, em cada instante, a posição de todos os carros na sua proximidade com um erro limitado. Só desta forma se conseguirá garantir que os carros tomem decisões seguras. Para se conseguir garantir esta percepção da realidade com um erro limitado, tem que se assegurar comunicação atempada. Infelizmente, é bastante difícil dar garantias de pontualidade em ambientes móveis sem fios.

O modelo Timely Computing Base (TCB) visa, precisamente, oferecer uma solução arquitectural que permita conciliar os requisitos de pontualidade das aplicações com os problemas da incerteza do ambiente. Um sistema com uma TCB dispõe de um conjunto de serviços que podem ser usados por um carro para detectar e reagir a tempo

*Faculdade de Ciências da Universidade de Lisboa. Bloco C5, Campo Grande, 1749-016 Lisboa, Portugal. Navigators Home Page: http://www.navigators.di.fc.ul.pt. Este trabalho foi parcialmente suportado pela Comissão Europeia, através do projecto IST-2000-26031 (CORTEX) e pela FCT, através do Laboratório de Sistemas Informáticos de Grande-Escala (LaSIGE).

a atrasos inesperados. Um carro pode assim conseguir parar antes que aconteça qualquer acidente - paragem segura - ou pode ainda adaptar-se continuamente ao ambiente - adaptação à Qualidade de Serviço (QdS) - para minimizar o número de paragens.

O artigo começa por descrever o modelo TCB de uma forma resumida na secção 2. Na secção 3 são descritos alguns dos problemas que enfrentámos na concretização da TCB num ambiente móvel sem fios e na secção 4 é apresentado um cenário de carros cooperantes que desenvolvemos recentemente usando essa concretização. Na secção 5 apresentamos uma avaliação preliminar do cenário concretizado. A secção 6 conclui o artigo.

2 Modelo TCB

O modelo TCB propõe uma aproximação arquitectural para lidar de uma forma genérica com o problema de programar aplicações com requisitos síncronos em sistemas com pontualidade incerta.

A figura 1 ilustra, segundo o modelo, a arquitectura de um sistema com uma TCB. Um sistema com uma TCB está dividido em duas partes bem definidas: parte *genérica* e parte de *controlo*. A parte genérica representa o que é normalmente chamado de "sistema" nas arquitecturas usuais. Funciona sobre uma rede global ou canal genérico (por exemplo, Internet) e é onde as aplicações executam e comunicam. A parte de controlo é constituída pelos módulos locais da TCB. Os módulos da TCB estão ligados por um canal de controlo. Os processos *P* representados na figura, executam em diferentes locais, fazendo uso da TCB sempre que seja apropriado. A parte do sistema correspondente à TCB, incluindo o canal de controlo, possui propriedades síncronas.

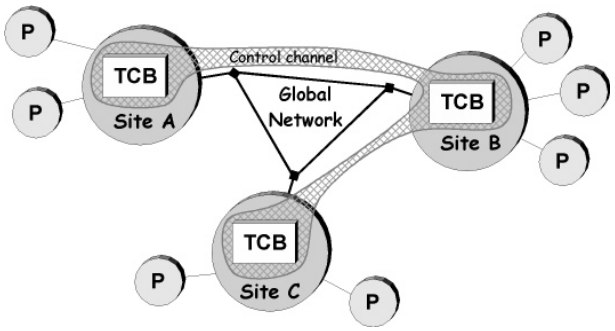

Figura 1: Arquitectura da TCB.

A TCB pode ser vista como um oráculo, utilizado pelas aplicações para resolver os seus problemas relacionados com tempo. Com este intuito, a TCB oferece serviços de suporte simples como, a capacidade de detectar falhas temporais, de medir durações e de executar atempadamente acções temporizadas. Enumeramos de seguida alguns dos serviços oferecidos pela TCB, com uma breve explicação dos mesmos acompanhada da respectiva assinatura:

Serviço de obtenção de estampilhas temporais Este serviço permite obter uma estampilha temporal gerada dentro da TCB. Todos os outros serviços da TCB requerem como argumento uma estampilha temporal que deve ser obtida previamente através deste serviço.

$estampilhaTemporal \leftarrow obterEstampilhaTemporal()$

Serviço de medições distribuídas Este serviço permite medir a duração de execuções distribuídas. Correspondendo a execução distribuída mais simples ao envio de uma mensagem, o serviço oferece uma API semelhante à dos sockets normais - $send()$, $receive()$ - e, no momento da recepção de uma mensagem, é retornada não só a informação recebida, mas também a duração da transmissão da mesma.

$send(estampilhaEnvio, ...)$

$duração \leftarrow receive(...)$

Serviço de detecção de falhas temporais Este serviço permite detectar se uma determinada acção termina fora do prazo estipulado. No caso de acções distribuídas, a API é uma extensão da do serviço de medições distribuídas: no momento de envio define-se adicionalmente qual o prazo para a entrega da informação e que função deve ser executada no caso de ser detectada uma falha temporal.

$send(estampilhaEnvio, prazo, handlerFalha, ...)$

$duração \leftarrow receive(...)$

A descrição detalhada de todos os serviços, assim como exemplos do seu uso podem ser encontrados em [13].

3 Concretização de uma TCB num ambiente móvel sem fios

O processo de concretização de uma TCB num ambiente móvel sem fios foi enquadrado por um conjunto de requisitos incontornáveis:

- Os dispositivos móveis que estavam disponíveis para a concretização eram *palmtops*[1] a correrem o sistema operativo Windows CE 3.0;
- O canal genérico, a ser utilizado pelas aplicações, deveria ser concretizado usando uma rede IEEE 802.11b[6] em modo ad-hoc;
- Necessidade de utilizar componentes de baixo custo (COTS[2]) e software aberto (não comercial).

Começámos por avaliar as características tempo-real do sistema operativo Windows CE 3.0. Em [3, 2] são referidas algumas dessas características, nomeadamente: suporte para interrupções encadeadas, bastantes níveis de prioridades de tarefas e escalonador preemptivo. Concluímos portanto que seria possível conseguir propriedades síncronas usando este sistema operativo.

Seguidamente, a questão que se colocou foi a de como concretizar fisicamente o canal de controlo com propriedades síncronas. Em seguida, apresentamos as várias hipóteses que surgiram, enumerando as vantagens e desvantagens de cada uma:

[1] IPAQ 3870 (`http://www.compaq.com/products/handhelds/pocketpc/H3870.html`)
[2] Commercial Off-The-Shelf

1. Partilhar o acesso ao meio (em termos práticos, partilhar a mesma placa de rede) com o canal genérico, usando um mecanismo para garantir os requisitos temporais do tráfego do canal de controlo. Um mecanismo deste tipo deve ser concretizado ao nível do MAC (*medium access control*) para garantir um comportamento tempo-real. Por exemplo, o protocolo TBMAC [9] constitui um bom exemplo de um protocolo MAC que permite garantir tempo de acesso ao meio limitado, numa rede ad-hoc sem fios. No entanto, dado não existirem ainda soluções de uso genérico que incorporem este, ou outros protocolos MAC determinísticos, considerou-se não ser viável a hipótese, na presente concretização, de usar um acesso ao meio partilhado;

2. Usar um acesso ao meio privado para o canal de controlo. Neste ponto, surgiram duas hipóteses em relação a que tecnologia utilizar para concretizar o acesso ao meio privado:

   Bluetooth[4] Dado que o *Bluetooth* estava disponível no nosso ambiente de desenvolvimento, esta tornou-se uma hipótese bastante atractiva. No entanto, verificou-se que o Windows CE 3.0 não suportava a criação de *sockets* deste tipo, estando essa funcionalidade disponível apenas nas versões mais recentes daquele sistema operativo - Windows CE 4.1 e seguintes. Desta forma, as únicas vias para utilizar a tecnologia *Bluetooth* seriam:

   (a) Usar uma biblioteca *Bluetooth* comercial, escolhida entre as várias existentes na Internet[1]. No entanto, devido ao elevado custo do software, esta solução não cumpria um dos requisitos enunciados;

   (b) Programar ao nível *série*, isto é, acedendo directamente ao driver *Bluetooth*. No entanto, a realização desta tarefa iria ter custos naturalmente elevados em termos de tempo de concretização, não sendo viável em face de outras possíveis soluções.

   IEEE 802.11b Uma outra hipótese seria utilizar uma rede privada IEEE 802.11b em modo ad-hoc como canal de controlo. Um dos problemas desta hipótese prendia-se com o facto de que estava definido, como referimos acima, que o canal genérico (aquele que seria utilizado pelas aplicações) seria também baseado em IEEE 802.11b. Desta forma, utilizando-se em simultâneo duas redes IEEE 802.11b, ter-se-ia que encontrar uma forma de não ocorrerem sobreposições ao nível das frequências de rádio utilizadas. O standard IEEE 802.11b permite 11 frequências na maior parte dos países da Europa[3], incluindo Portugal, as quais distam 5 Mhz entre elas. No entanto, uma transmissão afecta uma largura-de-banda de 30 Mhz. Desta forma, 2 frequências não se sobrepõem se estiverem distanciadas em mais do que 30 Mhz. Contudo, a concretização deste esquema pode ainda levantar outros problemas. Por exemplo, é necessário que as placas de rede (ou os respectivos *device drivers*) permitam a selecção do canal (frequência) a utilizar por omissão em modo ad-hoc. No caso do hardware que dispúnhamos, esta selecção era possível apenas numa das duas interfaces, o que ainda assim é suficiente para resolver o problema, desde que se saiba qual a frequência usada pela interface não configurável.

De acordo com os requisitos enunciados acima e considerando todas as hipóteses descritas, optámos por concretizar uma TCB sobre o sistema operativo Windows CE

[3] O número de frequências permitido varia entre continentes e por vezes entre países.

3.0, usando uma rede IEEE 802.11b em modo ad-hoc como canal de controlo[4]. Esta concretização consistiu em grande parte na adaptação de uma versão já existente da TCB para o sistema *Real-Time Linux*[7], na qual o canal de controlo é baseado numa rede privada com *Ethernet Comutada*. A concretização foi feita usando a linguagem de programação *C*.

Observe-se que o facto de termos uma rede privada IEEE 802.11b para o canal de controlo não é suficiente para que o mesmo tenha um comportamento síncrono. Dado que o modo de funcionamento da TCB obriga a que cada módulo local transmita periodicamente informação para todos os outros módulos, a inexistência de um controlo de acesso ao meio determinístico poderá levar à ocorrência de colisões entre os pacotes enviados pelos diferentes módulos locais. No IEEE 802.11b, cada pacote de dados enviado *ponto-a-ponto* com sucesso é confirmado por um pacote especial - ACK - enviado pelo destinatário ao emissor: um emissor que não receba um pacote ACK até um determinado prazo após o envio de um pacote de dados, retransmite este pacote. Quando um pacote de dados é enviado em *difusão*, não existe qualquer mecanismo de confirmação/retransmissão. Desta forma, uma colisão entre dois pacotes de dados enviados em difusão resulta na perca dos mesmos.

Tendo isto em conta, decidimos concretizar duas versões da TCB, que comparamos de seguida: $i)$ uma versão em que o envio em difusão é emulado por envios ponto-a-ponto para todos os nós da rede e $ii)$ uma versão em que é utilizado o envio em difusão normal.

Na versão $i)$ cada módulo local da TCB pode transmitir pacotes sempre que o pretenda. Aproveitamos neste caso o mecanismo de ACKs para mascarar eventuais omissões.

Na versão $ii)$, dado que temos de evitar as colisões, cada módulo local da TCB obedece a um protocolo de acesso ao meio do tipo "passagem de testemunho". Os módulos locais da TCB formam um anel lógico e existe um testemunho que o percorre. É claro que esta solução é comprometedora no que diz respeito à mobilidade, uma vez que exige o conhecimento da informação relativa aos processos que formam o anel, o que implicaria, no mínimo, a utilização de protocolos de gestão e manutenção desta informação. No entanto, para uma análise e avaliação meramente dos aspectos de tolerância a faltas e omissões da rede, consideramos legítimo assumir um cenário estático, sem falhas dos próprios nós. Cada módulo local da TCB só pode transmitir quando tem o testemunho e só pode transmitir um número máximo de bytes predefinido. Após a transmissão, o testemunho é passado ao próximo nó no anel lógico. Este modo de funcionamento é semelhante ao do Token Bus [5], sendo por isso possível obter determinismo.

A versão $ii)$ tem a grande desvantagem de deixar de funcionar se o testemunho se perde. No entanto, dado que o testemunho é representado por poucos bytes e é enviado ponto-a-ponto (beneficiando portanto do mecanismo de ACKs), a probabilidade de se perder é considerada reduzida.

Em termos de desempenho, um envio em difusão é obviamente mais escalável que um envio ponto-a-ponto para todos os nós. No entanto, para compararmos as duas versões da TCB, há a considerar também na versão $ii)$ o tempo de passagem do testemunho e o resultante tempo que cada nó espera antes de poder enviar uma mensagem. Dado que não dispúnhamos de módulos locais da TCB suficientes para poder averiguar sobre a escalabilidade de cada uma das versões, não podemos concluir

[4]Disponível para download em `http://www.navigators.di.fc.ul.pt/software/tcb/downloads.htm`

sobre qual teria melhor desempenho.

Em termos de faltas omissivas (perda de pacotes), a tabela 1 apresenta os resultados de alguns testes que efectuámos para averiguar o impacto da injecção contínua de pacotes de diferentes tamanhos no canal de controlo de cada versão da TCB. Os testes foram efectuados usando a ferramenta *ping* no modo *flood*[5] e demoraram 1 minuto para cada tamanho de pacote - o que corresponde ao envio de cerca de 6000 pacotes. Tendo em conta que o standard IEEE 802.11b define que um pacote de dados contém no máximo 2312 bytes, os tamanhos de pacote foram escolhidos de forma a analisar o impacto da fragmentação. Desta forma, os pacotes de 64 e 2000 bytes não sofrem qualquer fragmentação, ao contrário do pacote de 3000 bytes que necessita de ser fragmentado.

	Número de bytes injectados		
Versão da TCB	64 bytes	2000 bytes	3000 bytes
$i)$	OK	OK	OK
$ii)$	OK	OK	erro

Tabela 1: Impacto da injecção contínua de pacotes de diferentes tamanhos durante 1 minuto no canal de controlo de cada versão da TCB

Como se pode observar, a versão $i)$ revelou ser mais fiável, dado que consegue funcionar apesar da injecção contínua de pacotes de 3000 bytes, ao contrário do que acontece na versão $ii)$. Concluímos que a versão $ii)$ se comporta desta forma porque a colisão de um pacote enviado em difusão, com qualquer pacote que resulte de uma interferência externa, conduz automaticamente à perda dos pacotes envolvidos.

Por esta razão, associada às desvantagens da versão $ii)$ descritas atrás, optámos pela versão $i)$ uma vez que é aquela que oferece maior cobertura dos pressupostos de sincronia.

4 Cenário de carros cooperantes

4.1 Descrição

No cenário que concebemos existem apenas ruas perpendiculares. Cada carro cooperante circula numa rua diferente e avança sempre na mesma direcção. Os carros têm que se coordenar à medida que se aproximam de cada cruzamento. Dado que os cruzamentos não têm qualquer sinalização, aplica-se a regra de dar prioridade ao veículo que se apresenta pela direita.

4.2 Arquitectura

Cada carro é representado por um *palmtop*, cuja posição (que num cenário real seria fornecida por um mecanismo de localização como o GPS) é simulada, assim como a sua velocidade e direcção. Tal como se pode observar na figura 2, os carros comunicam usando placas de rede sem fios IEEE 802.11b em modo ad-hoc (uma para o canal genérico e outra para o canal de controlo da TCB). Para além dos *palmtops*, existe uma aplicação monitor executada num portátil, o qual tem também uma placa IEEE 802.11b

[5] envio de 100 pacotes por segundo

para comunicar com os carros. Esta aplicação monitor permite observar a “realidade”, ou seja, observar o comportamento dos carros e detectar quando acontecem acidentes.

Figura 2: Arquitectura de rede do cenário

Em cada *palmtop* corre uma TCB que concretizámos sobre o sistema operativo Windows CE 3.0 (ver secção 3).

A lógica de controlo de cada carro cooperante baseia-se na informação recebida sobre a localização dos outros carros e é executada por uma aplicação concretizada em Java. Esta aplicação tem acesso aos serviços da TCB através de uma biblioteca[6] também concretizada em Java.

A arquitectura dos vários componentes de software descritos é ilustrada na figura 3.

Figura 3: Arquitectura em cada *palmtop*

4.3 Mecanismo de falha segura

Neste cenário, cada carro transmite periodicamente um evento com a sua posição, velocidade e direcção para todos os outros carros, de forma a que cada um tente manter

[6] API está disponível em `http://lasige.di.fc.ul.pt/~pjsousa/tcb_javadoc/`

uma imagem tempo-real da realidade, isto é, uma imagem com um erro máximo bem delimitado. Para tal, os eventos têm que ser entregues atempadamente: a cada evento é associado um prazo de entrega que está relacionado com a velocidade do carro emissor (quanto maior a velocidade, menor o prazo de entrega) e esse evento deve ser entregue a todos os receptores dentro do prazo de entrega estipulado. É precisamente neste âmbito que é usada a TCB: se um carro detectar (através do serviço de detecção de falhas temporais da TCB) uma violação do prazo de entrega num evento que deveria receber, entrará num estado de falha segura, parando de imediato, isto é, antes de sofrer qualquer acidente.

Observe-se que quando é detectada uma violação do prazo de entrega de um evento, não é o carro emissor que entra num estado de falha segura, mas sim o carro receptor. Tal pode dar origem a que um carro que esteja a sofrer interferências de uma entidade exterior e cujos eventos se atrasam constantemente, provoque a paragem de todos os carros em seu redor. Não é um esquema justo, mas é um esquema seguro. Se permitíssemos que fosse o carro emissor a parar quando fosse detectada uma violação no prazo de entrega, poderiam acontecer acidentes.

Para comprovar este facto, considere-se o seguinte exemplo ilustrado na figura 4, em que o carro A e o carro B se aproximam de um cruzamento .

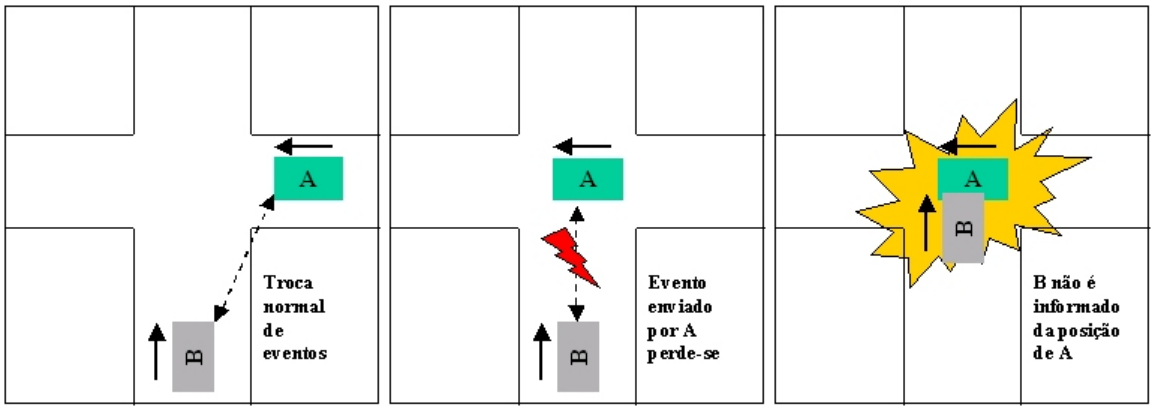

Figura 4: Exemplo do que acontece quando a detecção de uma violação no prazo de entrega de um evento apenas provoca a paragem do carro emissor.

No momento em que o carro A está a passar pelo cruzamento, envia um evento com a sua nova posição mas este perde-se. Quando A detecta a violação no prazo de entrega desse evento, entra num estado de falha segura, parando de imediato. No entanto, como o carro B não recebe o evento, não consegue saber que A está à sua frente parado e dá-se um acidente. Para agravar a situação, é o carro B que é declarado culpado, pois o carro A apresentava-se pela direita!

De um ponto de vista teórico, cada carro cooperante mantém uma base de dados com as posições de todos os outros carros, em que o valor da posição de cada carro tem uma validade associada. Sempre que essa validade seja ultrapassada, porque não se recebeu um novo evento desse carro com a (eventual) nova posição, a base de dados fica incoerente (com a realidade) e, consequentemente, o carro deve entrar atempadamente num modo de falha segura, parando, para evitar qualquer acidente.

4.4 Adaptação à QdS

Como se conclui da secção anterior, o mecanismo de falha segura é condição necessária e suficiente para que não ocorra qualquer acidente. No entanto, num cenário de carros cooperantes não podemos raciocinar apenas numa base de segurança. Uma das principais utilizações dos carros é o transporte de pessoas e queremos assegurar também o máximo conforto possível para as mesmas. Desta forma, um outro objectivo, para além do da segurança, será o de tentar garantir que os carros avancem a uma velocidade constante, havendo apenas variações suaves da mesma. Não queremos, desta forma, que um carro esteja num instante a avançar muito depressa, para, no instante seguinte, ser obrigado a parar porque o mecanismo de falha segura foi activado - resultado da detecção de uma violação no prazo de entrega de um evento que lhe era destinado.

Assim sendo, no nosso cenário, cada carro adapta a sua velocidade, em cada instante, à QdS providenciada pelo ambiente. O principal objectivo desta adaptação é evitar que o mecanismo de falha segura seja disparado.

A adaptação à QdS baseia-se num histórico de medições das latências dos eventos recebidos por cada carro. As latências são obtidas usando o serviço de medições distribuídas da TCB.

Tendo por base este histórico, é calculada a probabilidade com que cada latência se pode voltar a verificar no futuro. Mais concretamente, usando as fórmulas descritas em [8], é construída uma função de distribuição de probabilidade $p(l)$, em que para cada $l \in latências$, $p(l)$ representa a probabilidade de ocorrer uma latência inferior a l. A figura 5 apresenta uma função deste tipo. A título de exemplo, na função ilustrada a probabilidade de ocorrer uma latência inferior a 30.0 unidades de tempo (ut) é de cerca de 0.9.

Esta função evolui ao longo do tempo de acordo com o conteúdo do histórico, que reflecte a QdS permitida pelo ambiente.

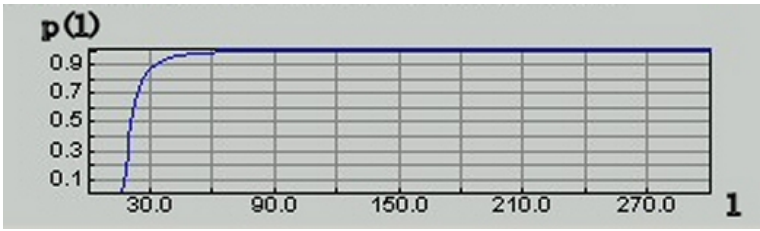

Figura 5: Exemplo da função de distribuição de probabilidade $p(l)$

Em cada instante, cada carro tem definida qual a probabilidade P de receber os eventos atempadamente (quanto maior o valor de P, menor a probabilidade do mecanismo de falha segura ser disparado). Usando esta informação, cada carro calcula analiticamente qual a latência l (associada à propagação dos eventos) tal que $p(l) = P$. Por exemplo, se num determinado instante um carro tiver definido $P = 0.9$, e se a sua função $p(l)$ corresponder àquela apresentada na figura 5, então a latência l resultante será 30.0 ut. A latência l calculada é depois usada para determinar, de acordo com uma tabela (latência, velocidade) que é comum a todos os carros, qual a velocidade a que o carro pode avançar.

Se um carro optar por manter sempre a mesma probabilidade P ao longo da sua operação, a sua velocidade aumentará ou diminuirá conforme o ambiente permita que a entrega dos eventos seja, respectivamente, mais rápida ou mais lenta.

5 Avaliação

O cenário de carros cooperantes descrito na secção anterior foi desenvolvido muito recentemente e portanto ainda não tivemos oportunidade de efectuar testes metódicos aos mecanismos de falha segura e de adaptação à QdS. Contudo, descrevemos nesta secção alguns testes preliminares ao mecanismo de falha segura.

A aplicação monitor, descrita na secção anterior, para além de permitir observar o comportamento dos carros, possibilita a injecção daquilo a que chamamos $nevoeiro\ eléctrico$ no canal genérico. Quando este $nevoeiro$ é injectado num determinado carro, este deixa de ver os outros carros a avançar. Mais concretamente, o $nevoeiro\ eléctrico$ é um injector de omissões.

A tabela 2 apresenta os resultados da injecção do $nevoeiro\ eléctrico$ com e sem o mecanismo de falha segura activado. Sem o mecanismo de falha segura activado, a injecção de $nevoeiro\ eléctrico$ provocou o acontecimento de acidentes. Com o mecanismo de falha segura activado, a injecção de $nevoeiro\ eléctrico$ nunca provocou qualquer acidente dado que os carros pararam sempre atempadamente quando detectaram violações dos prazos de entrega dos eventos que deveriam receber.

Estado do mecanismo de falha segura	Número de colisões
Desactivado	30
Activado	0

Tabela 2: Número de colisões de carros que se verificaram durante 5 minutos com injecção de $nevoeiro\ eléctrico$

O mecanismo actual de adaptação à QdS não tem em conta as omissões, apenas as latências dos eventos que são efectivamente entregues. Desta forma, não foi possível utilizar a injecção de $nevoeiro\ eléctrico$ para testar este mecanismo. Estamos neste momento a desenvolver um injector mais genérico de falhas temporais que permita injectar não só omissões, mas também atrasos nos eventos entregues.

6 Conclusão

Neste artigo descrevemos a construção de um cenário de carros cooperantes num ambiente móvel sem fios, que envolveu a concretização do modelo TCB sobre o sistema operativo Windows CE. Debatemos a necessidade de se usar de forma complementar os mecanismos de falha segura e de adaptação à QdS do ambiente, para se garantir segurança e conforto, respectivamente. Os testes que efectuámos permitem-nos concluir que os mecanismos funcionam como esperado, nomeadamente não se verificando acidentes, nem excessivas activações do mecanismo de falha segura.

Referências

[1] aveLink Bluetooth Stack. http://www.avelink.com/modules/bluetooth.htm.

[2] Updated with New Kernel Features, Windows CE 3.0 Really Packs a Punch. http://www.microsoft.com/msj/0799/wincekernel/wincekernel.aspx.

[3] Windows CE 3.0: Enhanced Real-Time Features Provide Sophisticated Thread Handling. http://msdn.microsoft.com/msdnmag/issues/1100/realce/default.aspx.

[4] Bluetooth sig. bluetooth technology. http://www.bluetooth.com/technology/, 1999.

[5] I. S. Association. IEEE Standard 802.4-1990, 1990.

[6] I. S. Association. IEEE Standard 802.11b-1999 (supplement to ANSI/IEEE Standard 802.11, 1999 edition), 1999.

[7] A. Casimiro, P. Martins, and P. Veríssimo. How to build a timely computing base using real-time linux. In *Proceedings of the 2000 IEEE International Workshop on Factory Communication Systems*, pages 127–1343, Porto, Portugal, Sept. 2000. IEEE Industrial Electronics Society.

[8] A. Casimiro and P. Veríssimo. Using the Timely Computing Base for dependable qos adaptation. In *Proceedings of the 20th IEEE Symposium on Reliable Distributed Systems*, pages 208–217, New Orleans, USA, Oct. 2001. IEEE Computer Society Press.

[9] R. Cunningham and V. Cahill. Time bounded medium access control for ad hoc networks. In *Proceedings of the Workshop on Principles of Mobile Computing (POMC 2002)*, Toulouse, France, Oct. 2002.

[10] P. Martins and Veríssimo. The Timely Computing Base and its future trends. In *Proceedings of the 7th CaberNet Radicals Workshop*, Bertinoro, Italy, Oct. 2002.

[11] P. Sousa and P. Veríssimo. Towards a cooperating autonomous car. In *Proceedings of the 7th CaberNet Radicals Workshop*, Bertinoro, Italy, Oct. 2002.

[12] P. Veríssimo and A. Casimiro. The Timely Computing Base model and architecture. *Transaction on Computers - Special Section on Asynchronous Real-Time Systems*, pages 916–930, Aug. 2002.

[13] P. Veríssimo, A. Casimiro, and C. Fetzer. The Timely Computing Base: Timely actions in the presence of uncertain timeliness. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 533–542, New York City, USA, June 2000. IEEE Computer Society Press.